Module-1

PAGE NO. :
DATE : / /

① What do you mean by pattern matching? Outline the Knuth-Morris-Pratt (KMP) algorithm and illustrate it to find the occurrences of the following pattern.

P: ABCDABD

S: ABC ABCDAB ABCDABC DABDE

A) Pattern matching is the process of identifying the occurrence of a specific pattern (p) within a larger body of data, or sequence (S).

The Knuth-Morris-Pratt (KMP) Algorithm is an efficient string-matching algorithm used to find the occurrences of a pattern within a text.

Steps of the Algorithm

1. Compute the Longest Prefix Suffix (LPS)
   - LPS[i] is the length of the longest prefix of the pattern that is also a suffix for the substring pattern [0..i]
2. Traverse the text & pattern
3. Compare characters one-by-one
4. Use the LPS array to skip unnecessary comparisons on a mismatch.

P = ABCDABD

S = ABCABCDABABCDABCDABDE

Step 1: Compute the LPS array.

M: lps[0] = 0

index i	character	Longest Prefix Suffix	lps[i]
0	A	-	0
1	B	-	0
2	C	-	0
3	D	-	0
4	A	A	1
5	B	AB	2
6	D	-	0

LPS array: [0,0,0,0,1,2,0]

Step 2: Search for Pattern P in Text S

2.1: Start comparing P with S at index 0

2.2: On mismatch, use the LPS array to determine the next alignment.

Text Index i	Match?	Next Alignment
0-6	No	Shift by 0
7-13	Yes	Continue
14-20	Yes	Continue

Output: P occurs at index 15 of S

② Write a function to evaluate the postfix expression. Illustrate the same for the given postfix expression

ABC-D*+E$F+G assume A=6, B=3, C=2, D=5, E=1, F=7

PAGE NO.:
DATE: / /

4)

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int top=-1, max=25, stack[25];

void push(int val){
    if (top >= max){
        printf("Overflow!");
    }
    else{
        top++;
        stack[top] == val;
    }
}
int pop(){
    int x;
    x = stack[top];
    top--;
    return x;
}
```

```
void main(){
    char *postfix;
    printf("Enter the postfix expression:");
    scanf("%c", &postfix);
    int i, len, top=-1;
    len = strlen(postfix);
    for (i=0; i<len; i++){
```

```
int evaluatePostfix(char* exp, int value[]){
    int i = 0;
    while (exp[i] != '\0'){
        if (isalpha(exp[i])){
            push(values[exp[i] - 'A']);
        }
        else{
            int b = pop(); a = pop();
            switch (exp[i]){
                case '+': push(a+b); break;
                case '-': push(a-b); break;
                case '*': push(a*b); break;
                case '/': push(a/b); break;
                case '$': push(pow(a,b)); break;
                default: printf("Invalid operator: %c\n",
                                exp[i]);
            }
        }
        i++
    } return pop();
}
```

```
int main(){
    char expression[] = "ABC-D*+E$+";
    int value[] = {6,3,2,5,1,7};
    int result = evaluatepostfix(expression, value);
    printf("Result of postfix evaluation: %d\n", result);
    return 0;
}
```

③ Write functions in C for the following operations without using built-in functions

i) Compare two strings

ii) Concatenate two strings

iii) Reverse a string

4) i)

```
int compare(char str1[], char str2[]){
    int i=0;
    if strlen(str1) != strlen(str2) {
        return 1;
    }
    while(str1[i] != '\0' || str2[i] != '\0'){
        if (str1[i] != str2[i]) {
            return 1;
        }
        i++;
    }
    return 0;
}
```

ii) void concatenate(char str1[], char str2[]){

```
int i=0, j=0;
while (str1[i] != '\0'){
    i++;
}
while (str2[j] != '\0'){
    str1[i] = str2[j];
    i++; j++;
}
str1[i] = '\0';
}
```

iii) void reverse(char str1[]){

~~int i=0; j = strlen(str1)~~

```
int i=0; j = strlen(str1)-1;
while (i<j){
    char temp = str1[i];
    str1[i] = str1[j];
    str1[j] = temp;
    i++; j++
}
}
```

Module-2

PAGE NO. :
DATE : / /

① Write the C function to add two polynomials. Show the linked representation of the below two polynomials and their addition using a circular singly link list.

P_1: $5x^3 + 4x^2 + 7x + 3$

P_2: $6x^2 + 5$

Output: add the above two polynomials & represent them using the linked list.

A)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct Node{
    int coeff;
    int pow;
    struct Node* next;
};

struct Node* createNode(int coeff, int pow){

    struct Node* new=(struct Node*)malloc(sizeof(Node));
    new->coeff=coeff;
    new->pow=pow;
    new->next=new;
    return new;

}
```

```
void addterm(struct Node** poly, int coeff, int pow){
    struct Node* new = createNode(coeff, pow);
    if (*poly == NULL){
        *poly = new;
        return;
    }
    struct Node * temp = *poly;
    while (temp->next != *poly){
        temp = temp->next;
    }
    temp->next = new;
    new->next = *poly;
}
struct Node* addpoly(struct Node* poly1, struct Node* poly2){
    struct Node *result = NULL, *p1 = poly1, *p2 = poly2;
    do{
        if (p1->pow == p2->pow){
            addterm(&result, p1->coeff + p2->coeff, p1->pow);
            p1 = p1->next;
            p2 = p2->next;
        } else if(p1->pow > p2->pow){
            addterm(&result, p1->coeff, p1->power);
            p1 = p1->next;
        } else {
            addterm(&result, p2->coeff, p2->pow);
            p2 = p2->next;
        }
    } while(p1 != poly1 && p2 != poly2);
```

```
    while (p1 != poly1){
        addterm(&result, p1->coeff, p1->pow);
        p1=p1->next;
    }
    while (p2 != poly2){
        addterm(&result, p2->coeff, p2->pow);
        p2=p2->next;
    }
    return result;
}

void display(struct Node* poly){
    struct Node* temp = poly;
    do{
        printf("%dx^%d", temp->coeff, temp->pow);
        temp = temp->next;
        if (temp != poly){
            printf("+");
        }
    }while (temp != poly);
    printf("\n");
}

int main(){
    struct Node *poly1=NULL, *poly2=NULL;
    addterm(&poly1, 5,3);
    addterm(&poly1, 4,2);
    addterm(&poly1, 7,1);
    addterm(&poly1, 3,0);
```

```
    addterm(&poly2, 6,2);
    addterm(&poly2, 5,0);
    printf("Polynomial 1:");
    display(poly1);
    printf("Polynomial 2:");
    display(poly2);
    struct Node* result = addpoly(poly1, poly2);
    printf("Result:");
    display(result);
    return 0;
}
```

Output:

Polynomial 1: $5x^3 + 4x^2 + 7x^1 + 3x^0$

Polynomial 2: $6x^2 + 5x^0$

Result: $5x^3 + 10x^2 + 7x^1 + 8x^0$

(2) Write a C program to implement insertion, deletion & display operations on a circular queue

PAGE NO. :
DATE : / /

A)

```
#include <stdio.h>
#define max 100
int queue[max], front=-1, rear=-1;
void enqueue(int n){
    if ((rear+1)%max == front){
        printf("Overflow!");
        return;
    }
    if (front == -1){
        front=0;
    }
    rear = (rear+1)%max;
    queue[rear]=n;
    printf("%d enqueued to the queue.\n",n);
}
void dequeue(){
    if (front == -1){
        printf("Underflow!\n");
        return;
    }
    printf("%d dequeued from queue.\n", queue[front]);
    if (front==rear){
        front=-1; rear=-1;
    }
```

```
    else {
        front = (front+1)%max;
    }
}

void display(){
    if (front==-1){
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements:");
    int i=front;
    while (i!=rear){
        printf("%d", queue[i]);
        i = (i+1)%max;
    }
    printf("%d\n", queue[rear]);
}

int main(){
    while(1){
        int choice,n;
        printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter digit to Enqueue:");
                    scanf("%d",&n);
                    enqueue(n);
                    break;
```

```
        case 2: dequeue();
                break;
        case 3: display();
                break;
        case 4: return 1;
        default: printf("Enter valid input !!!\n");
                break;
    }
}
return 0;
}
```

Module-3

PAGE NO. :
DATE : / /

① Define Sparse Matrix. For the given sparse matrix, give the linked list representation:

$$A=\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

A Sparse matrix is a matrix where most of the elements are zero(0).

- To conserve save space, only the non-zero elements are stored, along with their row & column indices. as a linked list.

Linked List Representation:

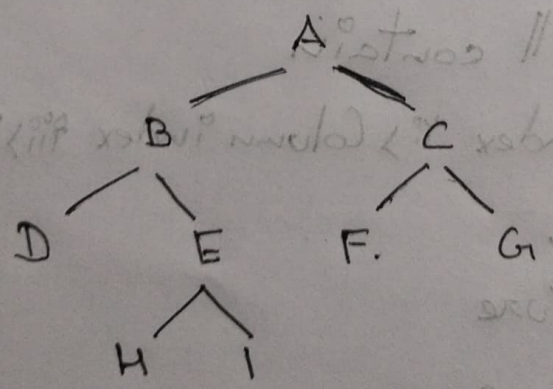
- Each node will contain
  i) Row index ii) Column index iii) Element value iv) Next pointer.
- Node structure

```
struct Node{
    int row, col;
    int value;
    struct Node* next;
};
```

- The linked list representation of the given matrix is

Row	Col	Value
0	2	3
0	4	4
1	2	5
1	3	7
3	1	2
3	2	6

(2) Write recursive C functions for inorder, preorder & postorder traversals of a binary tree. Also, find all the traversals for the given tree.



4)

```
int inorder(struct Node *root){
    if (root == NULL){
        return 0;
    }
    inorder(root->left);
    printf("%d", root->data);
    inorder(root->right);
}

int preorder(struct Node *root){
    if (root == NULL){
        return 0;
    }
    printf("%d", root->data);
    preorder(root->left);
    preorder(root->right);
}

int postorder(struct Node *root){
    if (root == NULL){
        return 0;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d", root->data);
}
```

Inorder

Inorder traversal of the given tree:

D B H E I A F C G

Preorder traversal of the given tree:

A B D E H I C F G

Postorder traversal of the given tree:

D H I E B F G C A

Module-4

PAGE NO. :
DATE : / /

(1) Define selection tree. Construct min winner tree for the runs of a game given below. Each runs consists of values of players. Find the first 5 winners

10	9	20	6	8	9	90	17
15	20	20	15	15	11	95	18
16	38	30	25	50	16	99	20
			28				

A) A selection tree is a binary tree used to find winners from multiple sorted lists or runs.
- The min winner tree identifies the smallest value among competitors in its nodes.

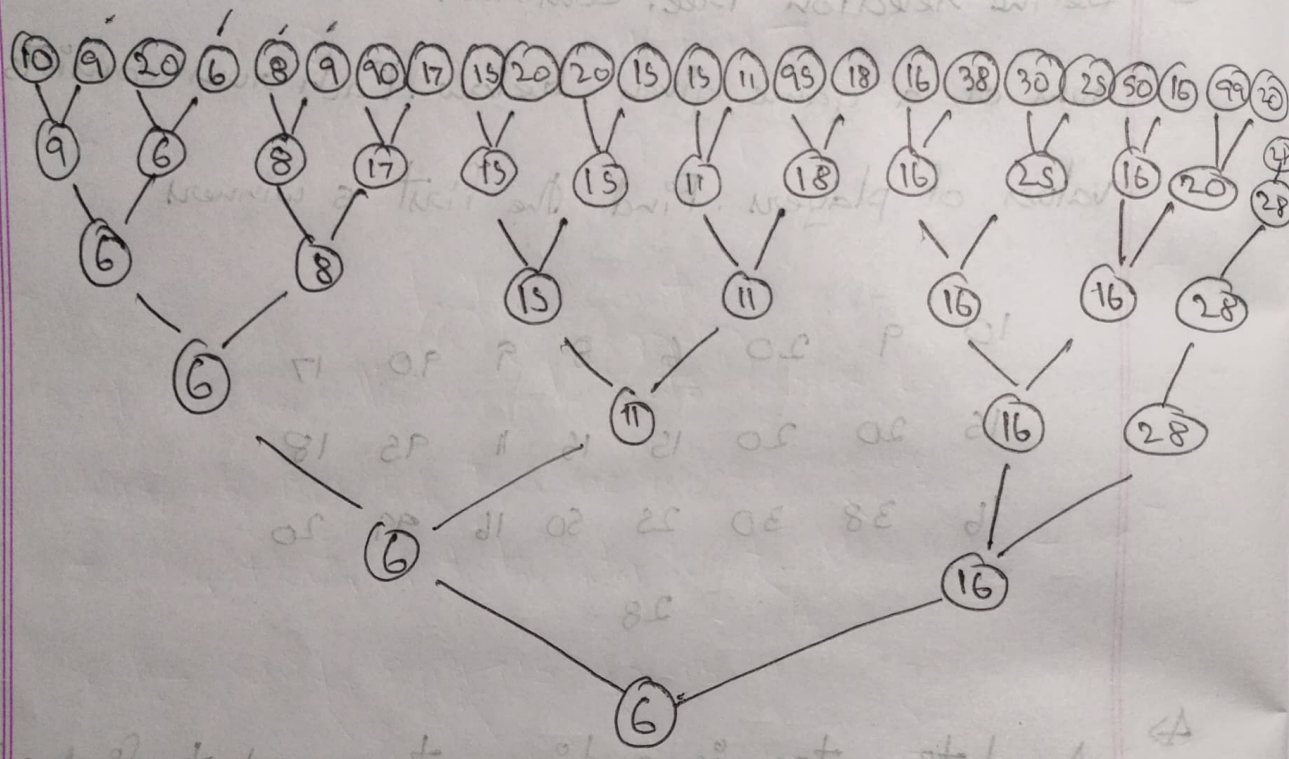
~~Cont~~
Steps to construct a min winner tree:

i) Represent each run as a ~~let~~ Leaf node.

ii) Compare two child nodes, storing the smaller value (winner) in the parent

iii) Repeat until the root node holds the smallest value.

- The min winner tree for the given values is:



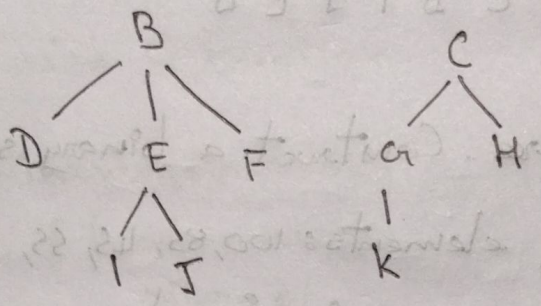
The winner is - 6.

The next winners are to be selected by reconstructing the tree by removing the winner from it.

∴ The first 5 winners are

6, 8, 9, 10, 11

(4) Define Forest. Transform the given forest into a Binary tree & traverse using inorder, preorder, & postorder traversal.



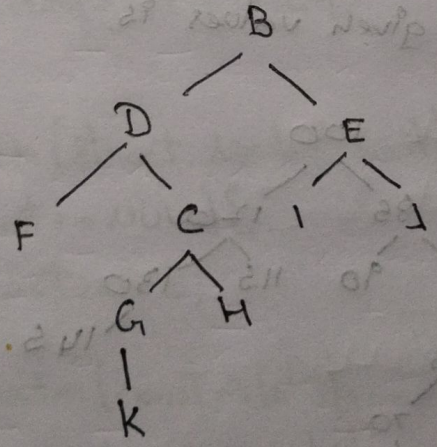
4) - A Forest is a collection of disjoint trees.

- To convert a forest to binary tree:

> Connect the left child of a node to its first child in the forest.

> Connect the right child of a node to its sibling in the forest.

- The binary tree for the given forest is



The traversal result for the binary tree is,

Inorder: F D K G C H B I E J

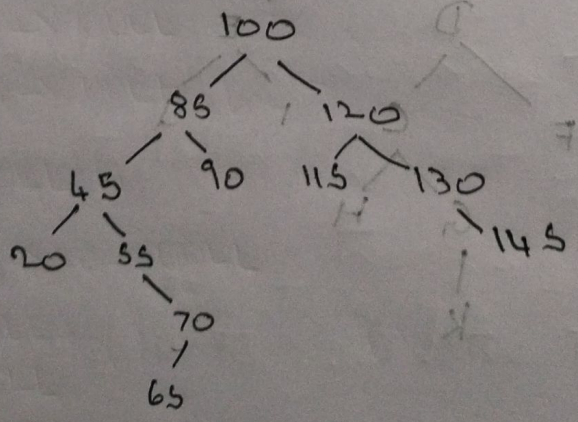
Preorder: B D F C G K H E I J

Postorder: F K G H C D I J E B

(3) Define Binary Search Tree. Construct a binary search tree (BST) for the following elements: 100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145. Traverse using inorder, pre-order & post-order traversal techniques. Write * recursive C functions for the same.

4) A Binary Search Tree (BST) is a binary tree in which the left subtree of a node contains only nodes with less value & the right subtree of a node contains nodes with greater value than it.

- The BST for the given values is,



- Inorder traversal : 20, 45, 55, 65, 70, 85, 90, 100, 115, 120, 130, 145

PAGE NO. :
DATE : / /

- Preorder traversal : 100, 85, 45, 20, 55, 70, 65, 90, 120, 115, 130, 145

- Postorder traversal : 20, 65, 70, 55, 45, 90, 85, 115, 145, 130, 120, 100

- Recursive C functions.

```
void inorder(struct Node* root){
    if (root==NULL){
        return 0;
    }
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
void preorder(struct Node* root){
    if (root == NULL){
        return 0;
    }
    printf("%d", root->data);
    preorder(root->left);
    preorder(root->right);
}
void postorder(struct Node* root){
    if(root==NULL){
        return 0;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d", root->data);
```

Module-5

PAGE NO. :
DATE : / /

① What is chained hashing? Discuss its pros & cons. Construct the hash table to insert the keys: 7, 24, 18, 52, 36, 54, 11, 23 in a chained hash table of 9 memory locations. ~~Use h(k)=~~ Use $h(k) = k \bmod m$.

4> - Chained Hashing is a collision-resolution technique in hash tables where each slot is a linked list. When multiple keys hash to the same index, they are stored in the linked list of that slot.

- Hash function: $h(k) = k \bmod m$,

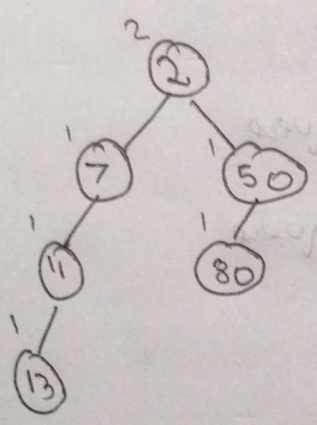
m -> size of the hash table.

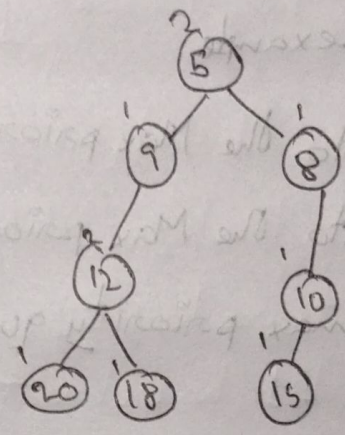
- Keys to enter: 7, 24, 18, 52, 36, 54, 11, 23.
- Table size: 9.

- The chained hash table is as follows.

Slot 0: NULL
Slot 1: 36 -> NULL
Slot 2: 11 -> NULL
Slot 3: NULL
Slot 4: 54 -> 18 -> NULL
Slot 5: NULL
Slot 6: NULL
Slot 7: 23 -> 7 -> NULL
Slot 8: 52 -> 24 -> NULL

(1) Define min Leftist tree. Meld the given min leftist trees.



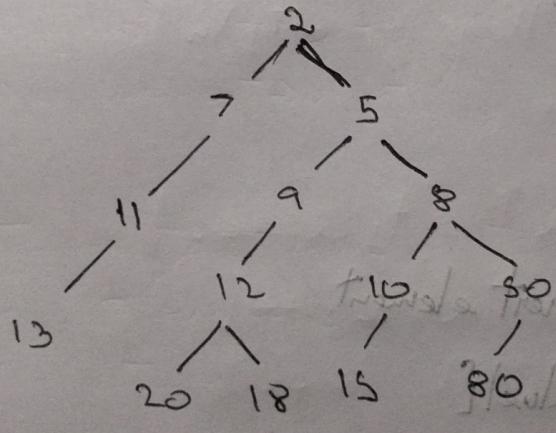


Ans) -A Min Leftist Tree is a binary tree where the following properties hold:

i) Every node has a key smaller than its children.

ii) The right subtree of every node is shorter than its left subtree.

-The melded min leftist tree is as follows.



③ What is a priority queue? Demonstrate Functions in C to implement the Max Priority queue with an example.

i) Insert into the Max priority queue
ii) Delete into the Max priority queue
iii) Display max priority queue.

4)

- Max Priority Queue is a data-structure where the highest priority element (largest value) is always at the front.

- ~~C prog~~ Demonstration with C program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int heap[MAX]
int size=0;

// Function to insert element.
void insert(int value){
    if (size >= MAX){
        printf("Heap is full.\n");
        return;
    }
```

```
int i = size;
heap[size++] = value;
while (i != 0 && heap[(i-1)/2] < heap[i]){
    int temp = heap[i];
    heap[i] = heap[(i+1)/2];
    heap[(i-1)/2] = temp;
    i = (i-1)/2;
}
}

int delete(){
if (size <= 0){
    printf("Heap is empty.\n");
    return -1;
}
if (size == 1){
    size--;
    return heap[0];
}
int root = heap[0];
heap[0] = heap[--size];
int i=0;
while (2*i+1 < size){
    int largest = i;
    int left = 2*i+1;
    int right = 2*i+2;
    if (left < size && heap[left] > heap[largest]) {
        largest = left;}
```

```
    if (right < size && heap[right] >
                        heap[largest]){
        largest = right;}

    if (largest == i){
        break;}

    int temp = heap[i];
    heap[i] = heap[largest];
    heap[largest] = temp;
    i = largest;
  }
  return root;
}

void display(){
    if (size == 0){
        printf("Heap is empty.\n");
        return;
    }
    printf("Max Priority Queue:");
    for (int i=0; i<size; i++){
        printf("%d", heap[i]);
    }
    printf("\n");
}
```

```
int main() {
    insert(10);
    insert(20);
    insert(15);
    insert(30);
    insert(25);
    printf("After inserting elements:\n");
    display();
    printf("Deleted max: %d\n", delete());
    display();
    printf("Deleted max: %d\n", delete());
    display();
    return 0;
}
```

Output:

After inserting elements:

Max Priority Queue: 30 25 15 10 20

Deleted max: 30

Max Priority Queue: 25 20 15 10

Deleted max: 25

Max Priority Queue: ~~20 15 10~~ 20 10 15